REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE 1 Aug 96	3. REPORT TYPE AND DATES COVERED Final Technical Report (1 Feb 93 - 31 Jan 96)

4. TITLE AND SUBTITLE Neural Network Approach Towards Logic Testing and Design for Testability	5. FUNDING NUMBERS F49620-93-1-0121 AFOSR-TR 96-0420
6. AUTHOR(S) Rai, Suresh	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Louisiana State University	8. PERFORMING ORGANIZATION REPORT NUMBER Tech Report EE 96-08-001
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Air Force Office of Scientific Research 110 Duncan Ave, Suite B115 Bolling AFB, DC 20332-8080 NM	10. SPONSORING/MONITORING AGENCY REPORT NUMBER
11. SUPPLEMENTARY NOTES	19960813 167
12a. DISTRIBUTION/AVAILABILITY STATEMENT Unlimited DISTRIBUTION STATEMENT A Approved for public release; Distribution Unlimited	12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

The report considers the problem of applying neural network for logic testing and proposes an efficient method based on the hyperneural model. The conventional Hopfield network of N neurons describes only binary relations between neurons. With this model, gates having more than two inputs need hidden neurons. Even two inputs XOR and XNOR gates require four neurons; one extra than that required by most other gates. Inclusion of an additional neuron doubles the search space. Thus, finding a valid test set using Hopfield model is either increasingly hard or the network converges to an invalid solution. The proposed hyperneural model overcomes these difficulties by using an energy function that not only considers binary relations but also captures all higher order relations between N neurons. A C++ code, developed for hyperneural network (HNN) based approach, is tested on a SUN SPARC 10/41 workstation for ISCAS '85 benchmark circuits and the results are compared with those obtained from MODEM and FAN. We have also applied the hyperneural concept for redundancy identification and removal problem in combinatorial circuits. Results, obtained for benchmark circuits, compare well with those given in the literature using conventional methods.

14. SUBJECT TERMS neural networks, hyperneural networks, Hopfield networks, logic testing	15. NUMBER OF PAGES 11
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT SAR

Air Force Office of Scientific Research grant F49620-93-1-0121

Grant duration: February 1, 1993 to January 31, 1996

(Final Report)

Report to the Air Force Office of Scientific Research

Neural Network Approach Towards Logic Testing and Design for Testability

Suresh Rai
(504) 388-4832
suresh@ee.lsu.edu

Technical Report EE 96-08-001
Department of Electrical and Computer Engineering
Louisiana State University
Baton Rouge, La 70803

ABSTRACT

The report considers the problem of applying neural network for logic testing and proposes an efficient method based on the *hyperneural* model. The conventional Hopfield network of *N* neurons describes only binary relations between neurons. With this model, gates having more than two inputs need *hidden* neurons. Even two inputs XOR and XNOR gates require four neurons; one extra than that required by most other gates. Inclusion of an additional neuron doubles the search space. Thus, finding a valid test set using Hopfield model is either increasingly hard or the network converges to an invalid solution. The proposed hyperneural model overcomes these difficulties by using an *energy function* that not only considers binary relations but also captures all higher order relations between *N* neurons. A C++ code, developed for *hyperneural network* (HNN) based approach, is tested on a SUN SPARC 10/41 workstation for ISCAS'85 benchmark circuits and the results are compared with those obtained from MODEM and FAN. We have also applied the hyperneural concept for redundancy identification and removal problem in combinational circuits. Results, obtained for benchmark circuits, compare well with those given in the literature using conventional methods.

INTRODUCTION

The ability to put large number of gates on a single chip of silicon offers great potential for reducing power, increasing speed, and lowering the cost of digital circuits. These advantages are, however, offset unless VLSI chips are economically tested. An obvious reason for testing a chip coming off the manufacturing line is to see if it works. Testing, or more precisely, the determination of what tests to apply to a particular device is a physical-design task requiring computer aid. An *automatic test pattern generation* (ATPG) is characterized as a search problem where the set of possible input vectors of a logic circuit is systematically searched until a test pattern is found. Note that this problem is known to be *NP*-complete.

We have begun investigating the ATPG problem using neural network based models because such models, described by a pseudo-Boolean quadratic function (called *energy function*), offer the advantage of applying optimization techniques. In addition, the *non-causal* form of the model allows the use of parallel processing for compute-intensive design automation tasks. By non-causal we mean that underlying models tend to blur the distinction between the input and output signals of logic elements. We also wanted to apply the method of our investigation for redundancy identification and removal problem in logic circuits, a sub-task commonly needed in synthesis problems. Besides the efforts of P. I., Suresh Rai, following graduate students helped achieve the goals of the project: W. Deng, K. Biswas, H. L. Johnson, R. Parthasarathy, Mohan D., Y. Liu, R. Jagdishan, A. Jagannnath, A. Bhosekar, A. Yellundur, and Xin Liao. While first two students were directly supported on the grant, remaining other students contributed by working on the topic or related issues as a part of their M.S. thesis requirements. A bibliography at the end of this report compiles our research results so far on the topic.

TECHNIQUE AND RESULTS

Theory

We, first, started investigating binary and three-valued *neural network* (NN) models proposed in the literature for ATPG problem. These methods use Hopfield's NN model or a variant of it to obtain an energy function for each 1- and 2-input gate (NOT, AND, OR, NAND, NOR, XOR). A collection of such functions is called a *basis* set. In the basis set, most two input gates require three neuron models. The XOR and XNOR gates, however, need four neurons to describe them. Neural models for all other gates with more than two inputs are constructed from the basis set by taking two inputs at a time. Since the underlying Hopfield network of N neurons describes only binary relations between them, gates with more than two inputs need hidden neurons to capture their functionality. Inclusion of an additional neuron in the model doubles the search space. Thus, having n extra neurons means the search space is increased by 2^n. This situation may lead to the problem of scaleability. It means with increasing problem size the network becomes so big that simulation times are excessively long. Also, because of the presence of state space explosion and non-linear nature of digital circuits, finding a valid test set becomes increasingly hard. Alternatively, it may happen that the search may end up in an invalid solution. An issue, relevant to this, is the *reliability* of result. Hopfield type neural networks are found to be particularly useful for problems where solutions close to the optimal are quite acceptable (character recognition, for example); in the test generation, a *close* solution does not mean much -- a vector is either a test or it is not. Another drawback of binary and three-valued NN models stems from the fact that they do not consider buffers and fan out lines. The basis set does not include any model for them; its 1-input NN list is limited to NOT gate only. Most circuits have both fan out lines and buffers in their description.

The proposed hyperneural network approach generalizes the energy function to capture binary and higher order relations among N neurons. A hypergraph is suggested to represent the modified energy function of HNN model. To define the hyperneural network, we use a fully-connected hypergraph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ consisting of a finite set $\mathcal{V}$ of vertices and a multiset $\mathcal{E} \subseteq 2^{\mathcal{V}}$ of *hyper edges* to represent the modified energy function

$$E = \left[K - \sum_{i}^{N} I_i V_i - \frac{1}{2!} \sum_{\substack{i,j \\ i \neq j}}^{N} T_{ij} V_i V_j \right] - \frac{1}{3!} \sum_{\substack{i,j,k \\ i \neq j \neq k}}^{N} T_{ijk} V_i V_j V_k - \ldots - \frac{1}{N!} \sum_{\substack{i,j,\ldots,l \\ i \neq j \neq \ldots \neq l}}^{N} T_{ij\ldots l} V_i V_j \ldots V_l \qquad (1)$$

Here K is a constant, and $N = |\mathcal{V}|$ is the number of neurons in the network. Various Ts are the weights associated with corresponding hyper edges. The elements of hyper edge $e \in \mathcal{E}$ are called its *terminals*. A hyper edge with two elements is also called an edge or *two-terminal hyper edge* and represents binary relation between neurons in the hyperneural network; a hyper edge with more than two terminals is sometimes called as *multi-terminal hyper edge* and depicts relations involving multiple neurons. In equation (1), the N summation term corresponds to N groups of hyper edges that have different number of terminals. The second summation term represents all the two terminal edges in the network while the ith summation term reflects all the hyper edges with i terminals. Let $\mathcal{E}^i$ denote the set of all hyper edges with i terminals then equation (1) can be rewritten as -

$$E = K - \sum_{i \in \mathcal{V}} I_i V_i - \sum_{i=1}^{N} \sum_{e \in \mathcal{E}^i} T_e \prod_{j \in e} V_j \qquad (2)$$

A hyper edge e is incident to the vertex v if $v \in e$. The degree of a vertex v is the number of hyper edges incident to v. We note that the Hopfield model represents a fully connected 2-graph where any neuron x connects any other neuron $y, x \neq y$, and $\{x, y\}$ is an edge of the graph. Like Hopfield network, we assume that the hypergraph representation of the hyperneural network is free from self-loops (i.e., terms like $T_{...i...i...} = 0$) and the graph is undirected. Note, the quadratic terms denoted by the expression within brackets in (1) are the same as Hopfield equation. By allowing extra (N-2) terms in (1), we introduce additional degrees of freedom at the cost of complicating the energy landscape of Hopfield network. However, for the purpose of modeling digital logic, the complicated energy landscape does not hurt because we deal with fixed weights on hyper edges. The additional degree of freedom provides the advantage of scaleability. Unlike Hopfield model, the HNN with N neurons can capture any relations between these N neurons.

For logic gates the weights can be obtained using the truth table of the block and applying *Mathematica* to solve equation (1). To illustrate, we consider a 3-input OR gate. Assigning energy function for the OR gate $E_{OR} = 0$ (1) for all the eight consistent (inconsistent) states of a 3-input OR gate, we get the 16 simultaneous equations listed in Table I which are solved for 16 unknowns as:

$$\begin{aligned} &K = 0;\ I_1 = I_2 = I_3 = I_4 = -T_{12} = -T_{13} = -T_{23} = T_{123} = -1\ ; \\ &-T_{14} = -T_{24} = -T_{34} = T_{124} = T_{134} = T_{234} = -T_{1234} = -2 \end{aligned} \tag{3}$$

An E_{OR} expression obtained using these parameters can be expressed as-

$$E_{OR} = -(1-V_4)\{2(1-V_1)(1-V_2)(1-V_3)-1\}+(1-V_1)(1-V_2)(1-V_3) \tag{4}$$

Figure 1 shows a hypergraph representation for E_{OR} given in (3). Here circles denote vertices of the hypergraph and dark triangles denote hyper edges with more than two terminals.

While we can find the energy function for all types of gates by solving the equation set, an alternative way to quickly get the energy function for all basic gates is to find the relation between the hyperneural networks for different gates. This formulation finds a non-linear pseudo-Boolean function for an n input gate directly from its logic description. To avoid confusion with the concept given in (1), we define the term as *maximum return function* (MRF), F_{gate}. (We have used $F_{gate} = -E_{gate}/a$, where $a \geq 1$.) Observe the following (in Table II)-

(a) the expression $(2V_1+2V_2-1)$ represents a threshold function for the OR or NOR gate,

(b) for the XOR or XNOR gate, the threshold function is $(2V_1+2V_2-4V_4-1)$, and

(c) a gate and its *single variable inversion* have the same maximum return function but for V_3 and $(1- V_3)$ factor.

To enhance the rate of convergence of a search technique, it is advisable to use scaling and change of variables if interaction between the variables can be eliminated . A function g(x) is said to be *non-interacting* if it is separable in its variables. In what follows, we discuss the substitutions for V_i's in Table II that help achieve variable separability and develop a theory for hyperneural networks. Let $V_1 = 1$, $V_2 = x_1x_2$, and $V_3 = y$ in F_{AND} where x_1, x_2, and y are binary variables. Then, $F_{AND} = y(2x_1x_2-1) - x_1x_2$. Substituting $V_1 = 0$, $V_2 = (1-x_1)(1-x_2)$, and $V_3 = (1-y)$, the MRF for an OR gate is obtained. These substitutions consider the enable inputs 1 (0) for AND and NAND (OR and NOR), and the algebraic function showing the functionality of a gate . For an XOR gate, $V_1 = x_1(1-x_2)$, $V_2 = (1-x_1)x_2$, $V_3 = y$, and $V_4 = 0$ will give

$$F_{XOR} = y[2x_1(1-2x_2)+2x_2-1]-[x_1(1-2x_2)+x_2] \tag{5}$$

To derive F_{XOR} from the MRF given in Table II, we have used $x^i \equiv x$ because $x \in \{0,1\}$. Here, $(1-2x_i)$ changes the logic $x_i = 0$ (1) to $x_i = 1$ (-1). Note the similarity between the F_{gate} expressions with (2).

Lemma 1 : A generalized expression for the maximum return function is given as

$$F_{gate} = z(2f-1)-f \tag{6}$$

where $f(z)$ is the function realized by (output of) the gate and is the right (left) side of functionality equations in Table II. ■

Lemma 1 is quite general as it helps obtain expressions for maximum return function for buffer, NOT, and fan-out lines as follows. Here m is the number of fan-out branches.

$$\begin{aligned} F_{buffer} &= y(2x-1)-x \\ F_{NOT} &= (1-y)(2x-1)-x \\ F_{fan-out} &= y_i(2x-1)-x \; ; \; \text{for } i=1,\ldots,m \end{aligned} \tag{7}$$

Lemma 2 : (a) For a consistent state, $F_{gate} = 0$.

(b) For an inconsistent state, F_{gate} is negative.

Proof. (a) When $z = f$, equation (6) gives $F_{gate} = 2f(f-1)$. Substituting $f = 0$ or 1 proves it.

(b) When $z \neq f$, then $z = 1$ (0) and $f = 0$ (1). In either case F_{gate} is negative ■

Lemma 3 : The consistent states of a gate is generated by maximizing $F_{gate} = -|z-f|$. ■

Method

There are three steps to generate test pattern for a given single stuck-at fault of a circuit under HNN model: circuit modeling, fault injection, and consistent state finding. Given a circuit, the maximum return function of each gate F_{gate} is first constructed using Lemma 3. The energy function of the circuit, ε_{ckt}, is the summation of the MRFs of all the gates.

$$\varepsilon_{ckt} = \sum_{\text{all the gates}} F_{gate} \tag{8}$$

A variable assignment satisfying that $\varepsilon_{ckt} = 0$ is a consistent state of the circuit. The set of all the consistent states is a logic specification of the circuit.

Circuit modeling constructs the energy function for a given circuit in a manner we discussed above. Fault is injected into the circuit by constructing a fault circuit from the original circuit, adding a XOR gate for each primary output to force the outputs from both circuits differing from each other at least one primary output. Precisely, these are done by -

1) For all the line i along the paths from the fault site to all primary outputs, assign an extra variable x_i'.
2) For all the gate along the paths from the fault site to all primary outputs, get the maximum return function for the gate in fault circuit F_{gate}'.
3) For each primary output y_i and its fault variable y_i', add a XOR gate. The outputs of all these XOR gates goes to an additional OR gate whose output is forced to be 1.
4) Sum all the MRFs getting from steps 2 and 3, and those describing the fault-free circuit. The result is the return function F_{gate}' for the fault injected model.
5) For the variable x_i and x_i' at the fault site, x_i' is assigned the fault value, and x_i' is assigned the opposite value to the fault.

The consistent states of the fault injected model satisfy F_{gate}' = 0. Since X = 0 and Y = 0 implies X+Y = 0, a solution for F_{gate}' = 0 is found by solving equation set $\mathcal{F}$= 0 for all $\mathcal{F}$ in F_{gate}'. To help achieve this, we use *satisfactionability* approach. We begin by assigning 0 (1) to one of the variables (using heuristics, given below), then simplify the equation set. If no contradiction (*viz.* 1 = 0) is found, we pick up another variable and assign a value to it. This is done repeatedly until a variable assignment that satisfies the equation set is found. In case of contradiction, we backtrack to a previously used variable. Following heuristics can be used to assign a variable:

a. Assign a logic value to a frequently occurring primary input or an internal line variable. This is achieved by obtaining a *weight* reflecting the number of times the variable occurs.

b. Consider only primary input variables for assignment using the descending order of their weights.

c. Assign primary inputs randomly.

d. Use simulated annealing to generate the input assignments.

Results, given below, are obtained with heuristic *b* which is a PODEM like strategy. It is different with the conventional PODEM in the sense that our HNN representation relies on a different theoretical foundation. By assigning logic values to neurons, we try to deterministically search for a zero energy state of the HNN *i.e.*, the solution for the equation set $\mathcal{F}$. Further, we apply a symbolic manipulation where some variables in the system are replaced by other variables. When these replacements occur, the variable-occurrence accounting is altered and the assignment ordering is, thus, dynamically changed.

Experimental Results

We coded our technique and tested it for Schneider, ECAT, C17, SN5483, and SN54181 circuits on the Sun SPARC 10/41 workstation. Table III summarizes the results using circuit specifications, testable and redundant faults, fault coverage, and average time needed to test the circuit. In all the cases, our method and Hopfield model obtain 100% fault coverage. However, based on average time per fault, HNN method outperforms the existing NN approach. In addition, we have obtained the number of aborted and redundant faults and fault coverage data for ISCAS'85 benchmark circuits. These results are shown in Tables IV and V, respectively. For comparison purposes, we have run MODEM (*module oriented decision making*) that is a deterministic ATPG approach based on PODEM but having extensions to improve performance. In Table V, we have borrowed results for PODEM and FAN. It is evident from Tables IV and V that our results are comparable to those obtained from some existing approaches. The difference in results can be attributed to two reasons. First, the symbolic strategy, used currently with the hyperneural network implementation, is responsible for its slack behavior. Second, a possible reason could be the absence of a sophisticated fault simulator. [Because of these reasons, the average time per fault in c432 through c1908 circuits is about 0.08 - 0.58 seconds, while in c2670 through c7552 it varies between 1- 4 seconds.] Note, an automatic test generation method is of practical use if the test pattern generation and fault simulation interact effectively. PODEM and FAN results are obtained using a modified concurrent fault simulator while MODEM uses *Mike* 2, a very fast state-of-the-art fault simulator approach .

Redundancy Identification and Removal

The r*edundancy identification and removal* (RID) is important not only in combinational circuits to help improve their testability but also in a combinatorial model such as a *fault tree with repeated events* (FTREs) that is used to perform dependability analysis in fault-tolerant systems. Our technique for RID using HNN employs two steps: First, the maximum return functions of all the gates in the circuit are formulated. Second, value assignment and simplification procedures are applied to arrive at the result. The proposed algorithm, implemented in C, is run on SPARC 10 machine for ISCAS'85 benchmark circuits. Tables VI provides the results. Reference [4] studies other techniques on the topic and gives a comparison with HNN based RID approach.

Other Related Research

We have also performed research on related topics such as CMOS testing, fault-tree analysis, evaluating signal probabilities in logic circuits, and fault simulation using pre-and post-processing. All these efforts were helpful in understanding different aspects of the ATPG problem.

BIBLIOGRAPHY

[1] S. Rai (1996), "Redundancy removal in combinational circuits using recursive learning", submitted to ESREL'97.

[2] S. Rai (1996), "A fast approach for estimating signal probabilities in general combinational circuits", in preparation.

[3] S. Rai and W. Deng (1996), "Hyperneural network - an efficient model for test generation in digital circuits", *IEEE Trans. Computers*, vol. 45, No. 1, January, pp. 115-121.

[4] A. Bhosekar, S. Rai, and R. Nath (1995), "Redundancy identification using hyperneural network", *6th Sym. on IC Tech., Systems, and Applications*, Singapore, Sept. 6 - 8, pp. 60 - 64. A. Bhosekar (M.S. thesis, 1995), Redundancy identification in combinational circuits using hyperneural network, Dept. of Electrical and Computer Engineering, L.S.U., Baton Rouge.

[5] A. Yellundur and S. Rai (1995), "Evaluating dependability models for fault-tolerant systems using Mathematica", *1st Int. Conf. on FTS*, I.I.T., Madras, India, Dec. 20 - 22, pp. 62-71.

[6] S. Rai (1995), "Evaluating FTREs for dependability measures in fault tolerant systems", *IEEE Trans. Computers*, vol. 44, No. 2, February, pp. 275 - 285.

[7] A. Jagannath and S. Rai (1995), "Impact of hardware and software faults on automatic repeat request schemes - an experimental study", *RAMS*, Washington DC, pp. 479 - 485.

[8] S. Rai (1994), "A direct approach to obtain tighter bounds for large fault trees with repeated events", *RAMS*, Anaheim, CA, pp. 475 - 480.

[9] R. Parthasarathy (M.S. thesis, 1994), New fault models for behavioral fault simulation in VHDL, Dept. of Electrical and Computer Engineering, L.S.U., Baton Rouge.

[10] M. Davuluri (M.S. thesis, 1994), Improved fault simulation using preprocess and postprocess techniques, Dept. of Electrical and Computer Engineering, L.S.U., Baton Rouge.

[11] K. Biswas and S. Rai (1994), "Testable realization of CMOS combinational circuits for voltage and current testing", *7th Int. conf. on VLSI Design*, Calcutta, India, pp. 197 - 202.

[12] S. Rai, H. L. Johnson, and V. Ratnam (1993), "NEUDEM: Neural network based decision making for generating tests in digital circuits", *36th Midwest Symp. on Circuits and Systems*, Detroit, pp. 596-599.

[13] R. Jagdishan (M.S. thesis, 1993), Estimating signal probabilities for testability measurements in general combinational circuits, Dept. of Electrical and Computer Engineering, L.S.U., Baton Rouge.

[14] Y. Liu (M.S. thesis, 1993), Conservative assessment of complex fault tree models, Dept. of Electrical and Computer Engineering, L.S.U., Baton Rouge.

[15] K. Biswas (M.S. thesis, 1993), Logic testing of CMOS VLSI circuits, Dept. of Electrical and Computer Engineering, L.S.U., Baton Rouge.

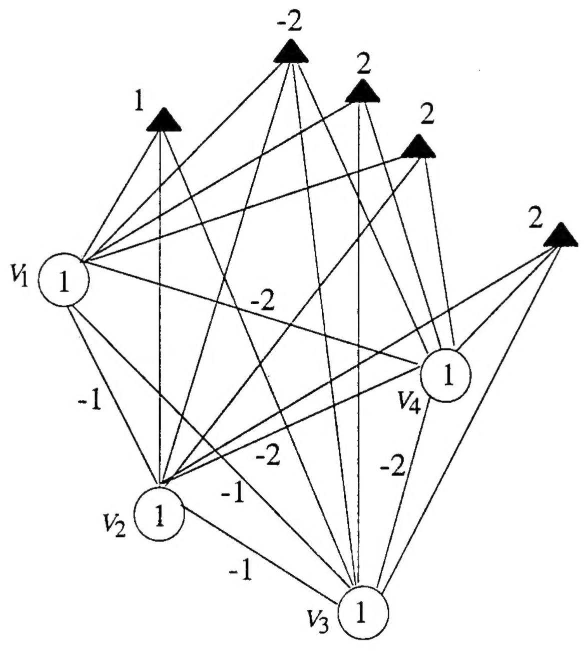

Figure 1

Table I

V_1	V_2	V_3	V_4	E_{OR} Expression using (1)	E_{OR} Value
0	0	0	0	K	0
0	0	0	1	$K - I_4$	1
0	0	1	0	$K - I_3$	1
0	0	1	1	$K - I_3 - I_4 - T_{34}$	0
0	1	0	0	$K - I_2$	1
0	1	0	1	$K - I_2 - I_4 - T_{24}$	0
0	1	1	0	$K - I_2 - I_3 - T_{23}$	1
0	1	1	1	$K - I_2 - I_3 - I_4 - T_{23} - T_{24} - T_{34} - T_{234}$	0
1	0	0	0	$K - I_1$	1
1	0	0	1	$K - I_1 - I_4 - T_{14}$	0
1	0	1	0	$K - I_1 - I_3 - T_{13}$	1
1	0	1	1	$K - I_1 - I_3 - I_4 - T_{13} - T_{14} - T_{34} - T_{134}$	0
1	1	0	0	$K - I_1 - I_2 - T_{12}$	1
1	1	0	1	$K - I_1 - I_2 - I_4 - T_{12} - T_{14} - T_{24} - T_{124}$	0
1	1	1	0	$K - I_1 - I_2 - I_3 - T_{12} - T_{13} - T_{23} - T_{123}$	1
1	1	1	1	$K - I_1 - I_2 - I_3 - I_4 - T_{12} - T_{13} - T_{14} - T_{23} - T_{24} - T_{34} - T_{123} - T_{124} - T_{134} - T_{234} - T_{1234}$	0

Table IV

Circuit	C432	c499	c880	c1355	c1908	c2670	c3540	c5315	c6288	c7552
No. of faults	524	758	942	1574	1879	2747	3428	5350	7744	7550
MODEM	4	8	0	10	12	129	210	144	348	757
HNN	4	0	42	0	15	212	311	83	32	312

Note : The backtrack parameter is selected as 10 in both MODEM and HNN techniques.

Table II

Gate	Maximum Return Function, F_{gate}	Output Function
AND	$V_3(2V_1+2V_2-3)-V_1V_2$	$y = x_1x_2$
NAND	$(1-V_3)(2V_1+2V_2-3)-V_1V_2$	$1-y = x_1x_2$
OR	$V_3(2V_1+2V_2-1)-(V_1+V_2+V_1V_2)$	$1-y = (1-x_1)(1-x_2)$
NOR	$(1-V_3)(2V_1+2V_2-1)-(V_1+V_2+V_1V_2)$	$y = (1-x_1)(1-x_2)$
XOR	$V_3(2V_1+2V_2-4V_4-1)-(V_1+V_2+6V_4+2V_1V_2-5V_1V_4-5V_2V_4)$	$y = x_1(1-2x_2)+x_2$
XNOR	$(1-V_3)(2V_1+2V_2-4V_4-1)-(V_1+V_2+6V_4+2V_1V_2-5V_1V_4-5V_2V_4)$	$1-y = x_1(1-2x_2)+x_2$

Table III

Circuits	Schneider	ECAT	C17	SN5483	SN54LS181
# of Inputs	4	6	4	9	14
# of Outputs	3	1	2	5	8
# of Gates	8	9	6	36	65
# of Faults	56	46	34	208	388
Redundant Faults	2	8	0	0	0
Testable Faults	54	46	34	208	388
Fault Coverage	100%	100%	100%	100%	100%
Average Time per fault (sec)	0.001	0.0027	0.007	.005	0.0178
*Average Time per fault	0.12	0.43	-	3.3	-

* The results help compare our approach with NN based technique.

Table V

Benchmark	Circuit	C432	c499	c880	c1355	c1908	c2670	c3540	c5315	c6288	c7552
% Tested Faults	PODEM	91.5	99.4	100	99.5	99.5	94.6	95.5	98.3	99.5	96.8
	MODEM	99.43	98.94	100	99.36	99.63	96.18	95.30	98.06	95.92	90.66
	FAN	93.7	97.2	100	97.5	99.3	95.7	95.8	98.9	99.4	98.2
	HNN	99.43	100	95.54	100	99.36	94.76	93.64	99.53	100	96.48
Average time per fault using HNN (sec)		0.23	0.08	0.42	0.33	0.58	2.82	2.66	2.81	1.01	4.04

Note : The backtrack parameter is selected as 10 in all the techniques.

Table VI

Benchmark circuits		C17	C432	C880	C1355	C1908	C2670	C3540	C5315
Number of stems		3	89	125	259	385	454	579	806
Number of Reconvergence points		2	92	113	394	250	357	600	799
Number of Redundancies identified	[A]	-	-	-	-	4	29	93	20
	[B]	0	4	0	8	9	117	137	59
	[C]	0	0	0	0	8	40	124	22
	[D]	0	0	0	0	8	41	136	23
Total CPU time, sec	with D	0	0.59	1,94	7.93	9.13	15.19	40.63	67.68

Note: [A] is Iyer and Abramovici's paper in *Int. Conf. on VLSI Design*, Jan 1994, p315; [B] means Schulz and Auth 's paper in *IEEE CAD*, July 1989; [C] & [D] refer to two versions of the proposed HNN based algorithm.